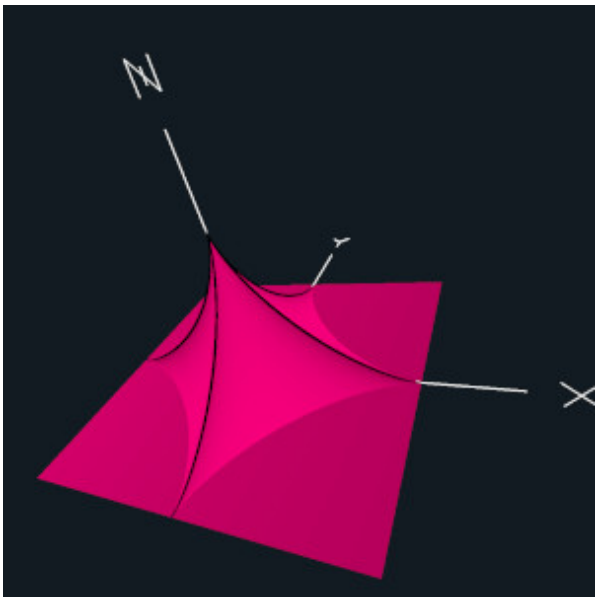
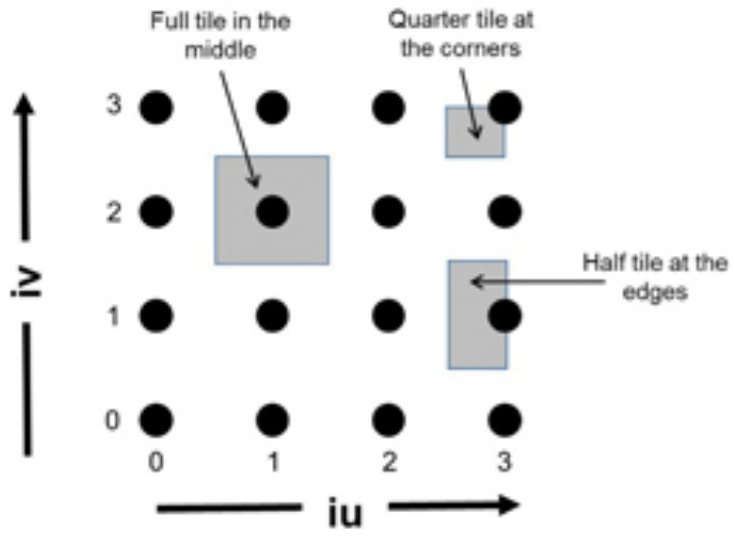
Project #2, Numeric Integration
Vincent Rastello
rastellv@oregonstate.edu

Project 1 Summary:

I will be integrating the area under a superquadratic surface with N = 0.7. This will be done by dividing up the area under the shape into 2D tiles and taking the heights of those tiles. See below diagrams.





The volume under the shape will then be calculated using parallel for loops to sum up the heights. The performance will then be recorded as Mega Heights computed per second. I will then graph and investigate the performance increase, f parallel and max speed up for this problem.

Results:

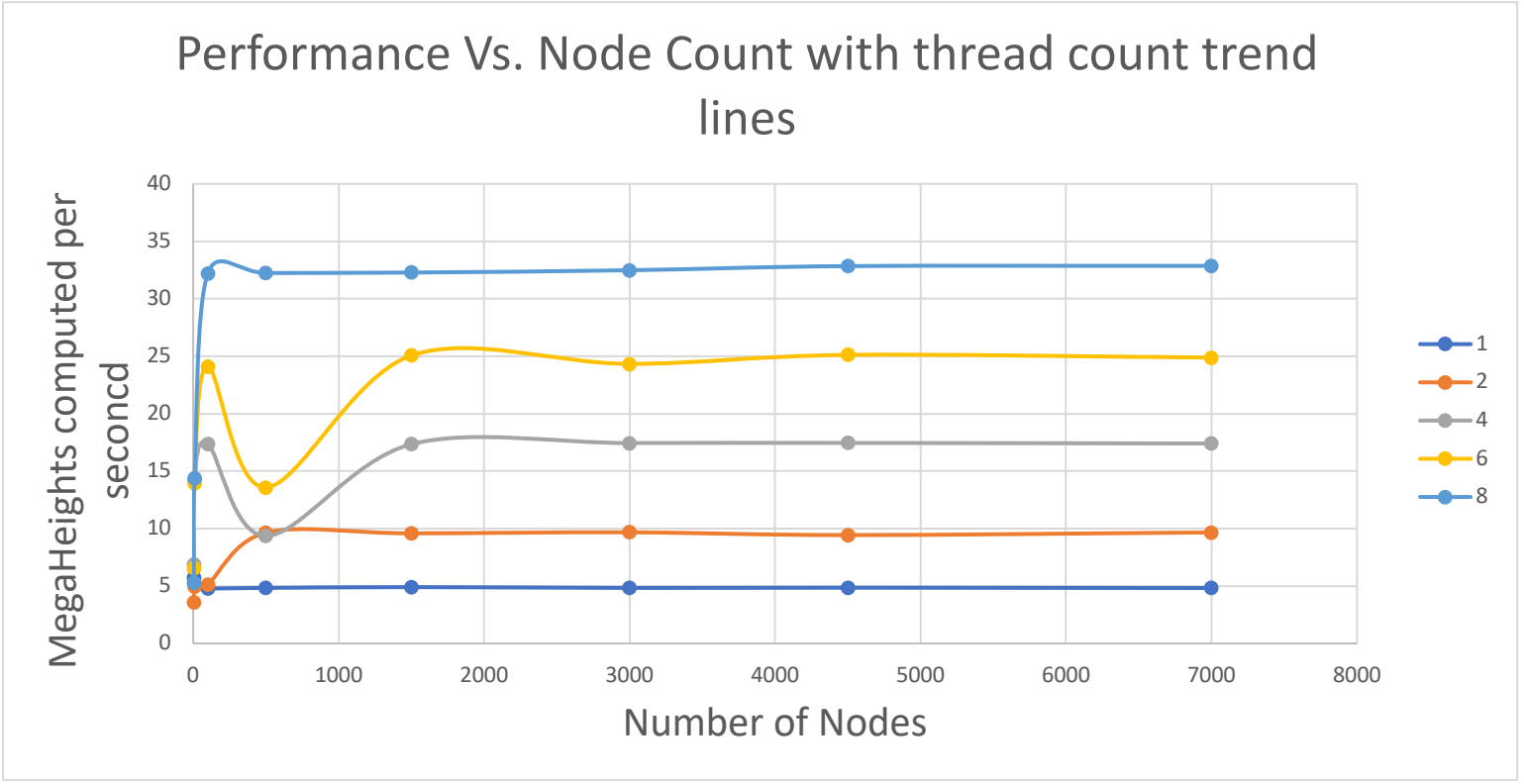
*Values in table are Mega Heights computed per second

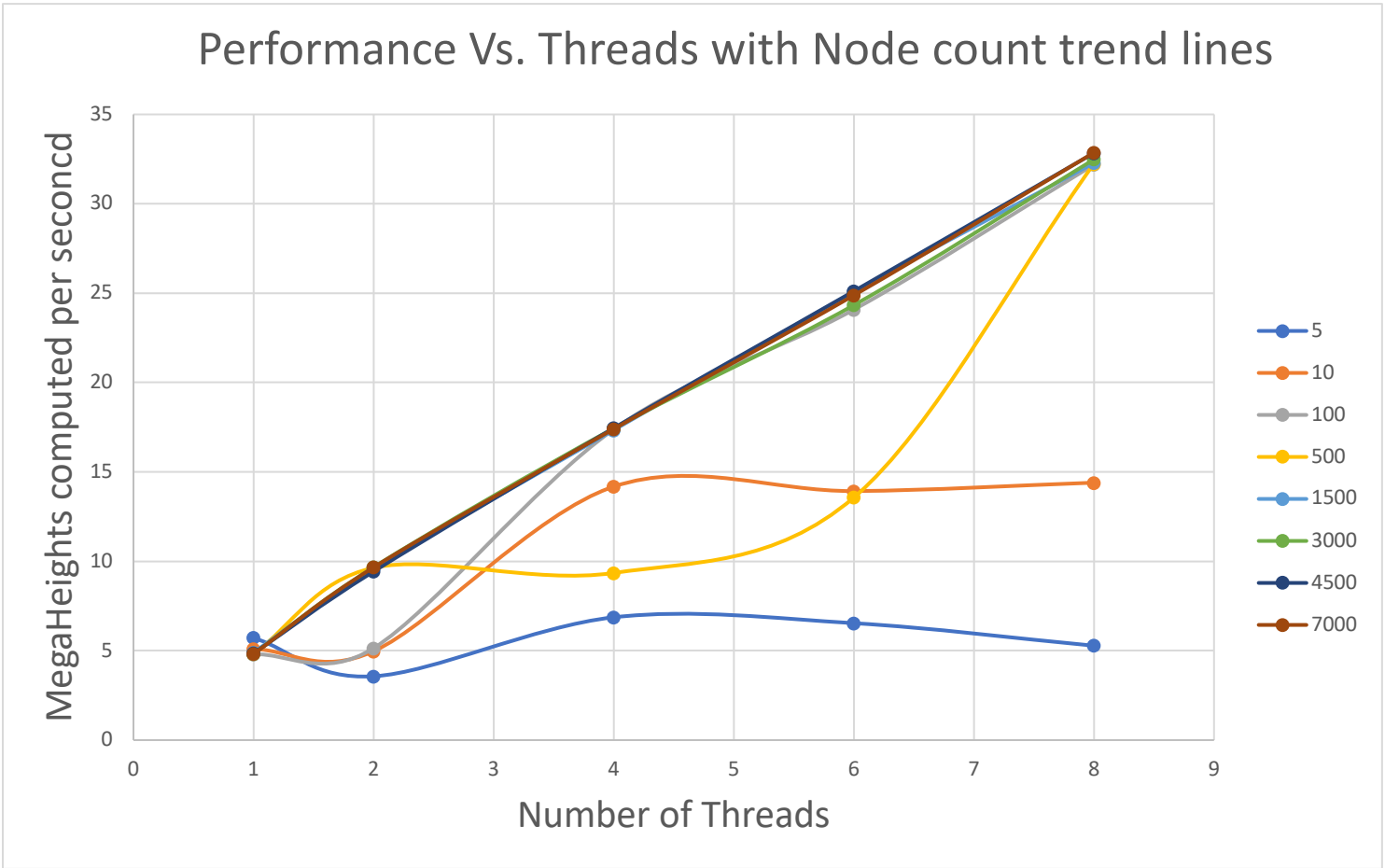
				Number of Nodes					
		5	10	100	500	1500	3000	4500	7000
	1	5.72	5.1	4.8	4.83	4.9	4.83	4.85	4.82
Threads	2	3.57	4.97	5.14	9.64	9.58	9.67	9.43	9.66
	4	6.87	14.18	17.38	9.35	17.34	17.44	17.45	17.4
	6	6.54	13.93	24.08	13.57	25.06	24.33	25.11	24.88
	8	5.28	14.4	32.19	32.24	32.28	32.48	32.84	32.86

1). This was ran on the OSU linux server with the following uptime: **83 users, load average: 4.55, 4.88, 4.72**
2). The actual volume integrated:

- 8 threads : 7000 nodes ; **volume = 0.435781480468**

Project #2, Numeric Integration
Vincent Rastello
rastellv@oregonstate.edu





Vincent Rastello
rastellv@oregonstate.edu

4). This data is showing us mostly what we would expect from parallelism, as the thread count goes up the performance goes up. After 1500 nodes the performance balances out, each thread count with a lower ceiling. As node count increases we see a more linear curve of performance boosts as number of threads goes up. Node count 5-500 have ups and downs in performance, with node counts 5 and 10 showing asymptotic curves. There are also dips in performance at the 100-500 node range with 4-6 threads.

5).Performance is going up due to thread count because you are utilizing multiple threads to perform the same tasks in parallel which achieves a greater rate of computations per second. For this problem, 1500 nodes seems to be the ceiling at which you cannot increase performance for the given thread count. This is because there are enough tasks for the threads to divide up efficiently, lower than 1500 and the collisions/waiting due to reduction will get in the way of performance. Anything higher than 1500 nullifies the collisions and the performance plateaus for higher node counts. At the lower node count ranges (5, 10), we see asymptotic curves as number of threads increases, at node count 5 this makes sense because after thread count 4 there are more threads than tasks, so some threads are idling. At 10 you see a higher performance but the same asymptotic, I'm not sure why it plateaus after 4, you would expect it to keep rising up to 8 as there are 10 loops to work through. This may be due to the load times of the OSU server. The ups and downs and dips in performance at the 100-500 node range at 4-6 threads may be due to collisions/idling due to reduction or the load times of the OSU server.

6).
*Fp (F Parrallel) Calculation:

*For values I will be using the 7000 node run.

n (number of threads) = 8
s (speedup) = (Performance with 8 threads)/(Performance with 1 thread)

$$F_p = \frac{n}{n-1}\left(1 - \frac{1}{\frac{MegaHeightsPerSec_n}{MegaHeightsPerSec_1}}\right)$$

$$= \frac{n}{n-1}\left(1 - \frac{MegaHeightsPerSec_1}{MegaHeightsPerSec_n}\right)$$

$$= \frac{8}{7}\left(1 - \frac{4.82}{32.86}\right)$$

$$\boldsymbol{F_p = 97.52\%}$$

7) Max Speed up Calculation:

maxSpeedup = 1/(1-fp) = **40**